Contenido

Variables locales

Muchas veces debemos hacer uso de variables locales para implementar estructuras de control de flujo con bucles.

Las variables locales sólo tienen vigencia mientras la rutina está en ejecución. Son variables auxiliares que necesitamos para almacenar valores intermedios con el fin de implementar el algoritmo (conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.).

Recordemos cómo se crean. Las variables locales (internas) en una rutina es necesario declararlas:

```
DECLARE var_name[,...] tipo [ valor PREDETERMINADO ]
```

Este comando se usa para declarar variables locales. Para proporcionar un valor por defecto de la variable, podemos incluir una cláusula DEFAULT
El valor puede especificarse como expresión, no necesita ser una constante. Si la cláusula DEFAULT no está presente, el valor inicial es NULL.
La visibilidad de una variable local está dentro del blog BEGIN ... END donde está declarado. Si hay blog DECLARE, debe ir inmediatamente después de COMENZAR.

> Nota: Antes de usar una variable en un bucle, es necesario inicializarla (asignarle un valor). Si no, el valor de la variable es NULL. Si el valor de una variable es NULL, cualquier expresión de la que forme parte tendrá valor NULL.

Constructores de control de flujo

Cuando se crea una rutina (función o procedimiento), todas las sentencias se ejecutan siguiendo una secuencia, es decir, el flujo de ejecución del programa consiste en ejecutar una instrucción detrás por otra. Las estructuras de control sirven para controlar ese flujo del programa, la toma de decisiones y programar las acciones en consecuencia.
Todas las estructuras de control están basadas en la evaluación de una condición más o menos compleja. En caso de cumplirse la condición, la evaluación de como resultado TRUE, se ejecutará un blog de sentencias, en caso negativo otro blog o nada. Los constructores pueden estar anidados.

Hay dos tipos de sentencias de control de flujo:

- Condicionales. IF y CASE. En las condicionales, la decisión que se toma es alternativa, es decir, se ejecuta un bloque de sentencias u otro.
- Iterativa (bucles). WHILE y REPEAT. La decisión que se toma es si repetir la ejecución de un blog de sentencias o no repetirlo.

Sentencia IF

```
SI condición_de_búsqueda ENTONCES lista_de_declaraciones
  [ELSEIF condición_de_búsqueda THEN lista_de_declaraciones] ...
  [Lista de declaraciones ELSE ]
FIN SI
```

Por ejemplo: Algoritmo usando IF que dado un número entero que denominaremos num asigna a la variable resultado el valor 'POSITIVO', 'NEGATIVO' o 'CERO' dependiendo del número.

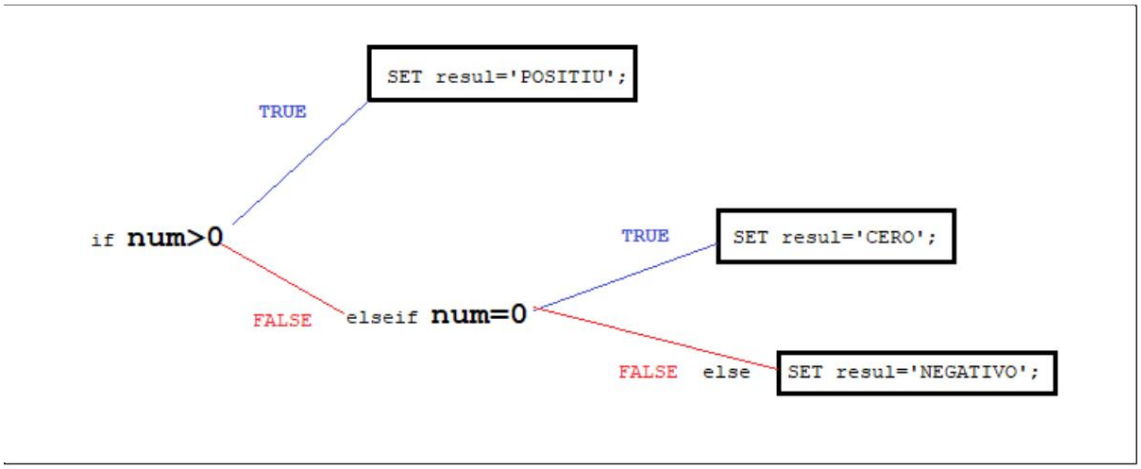
Podría ser:

```
si num>0 entonces
        SET resul='POSITIVO';

-- Este blog sólo se ejecuta si la condición del IF se evalúa a
FALSO

de lo contrario, si num=0 entonces --
        ESTABLECER resultado='CERO';
    demás
        SET resul='NEGATIVO';
fin si;
```

Un IF es como un árbol de decisión donde dependiendo de si la condición es verdadera o falsa se realiza un blog de código u otro. Podemos representar el algoritmo anterior así:



Para probarlo, deberíamos incluirlo dentro de una rutina: procedimiento o función. Por ejemplo, en el siguiente procedimiento:

```
delimitador //
crear procedimiento ejemplo1(num double, OUT resul varchar(10))
comenzar
        si num>0 entonces
                SET resul='POSITIVO';
        de lo contrario, si num=0 entonces
                ESTABLECER resultado='CERO';
        demás
                SET resul='NEGATIVO';
        fin si;

fin //

/*Probamos el funcionamiento*/
delimitador;
LLAMAR ejemplo1(5,@h);
SELECCIONAR @h;
```

También obtenemos el mismo resultado con ese otro algoritmo:

```
        si num>0 entonces
                SET resul='POSITIVO';
        Fin si;

        si num=0 entonces
                ESTABLECER resultado='CERO';
        Fin si;

        si num<0 entonces
                SET resul='NEGATIVO';
        Fin si;
```

Sentencia CASE

En MySQL existen dos formatos para la sentencia CASE.

```
CASO
    CUANDO condición_de_búsqueda ENTONCES lista_de_declaraciones
    [CUANDO condición_de_búsqueda ENTONCES lista_de_declaraciones] ...
    [Lista de declaraciones ELSE ]
CASO FINAL
```

Por ejemplo, para el ejercicio anterior con CASE sería:

```
CASO
        CUANDO num>0 entonces
                SET resul='POSITIVO';
        CUANDO num=0 entonces
                ESTABLECER resultado='CERO';
        CUANDO num<0 entonces
                SET resul='NEGATIVO';
Fin del CASO;
```

O también:

```
CASO
        CUANDO num>0 entonces
                SET resul='POSITIVO';
        CUANDO num=0 entonces
                ESTABLECER resultado='CERO';
        DEMÁS
                SET resul='NEGATIVO';
Fin del CASO;
```

Esta otra sintaxis sólo permite la comparación de igualdad entre una variable () y las posibilidades de coincidir con los valores descritos en alguno de los WHEN.

```
CASO valor_del_caso
  CUANDO cuando_valor ENTONCES lista_de_declaraciones [CUANDO
  cuando_valor ENTONCES lista_de_declaraciones] ... [SINO
  lista_de_declaraciones]
CASO FINAL
```

https://dev.mysql.com/doc/refman/5.7/en/case.html

Sentencia REPEAT

En todos los bucles debe tenerse en cuenta que dentro del bloque de sentencias debe haber al menos una que permita que la evaluación de la condición cambie en algún momento (cambie el valor de la variable(s) sobre el que está construida la condición). Si no, nos encontraríamos ante un bucle infinito.

Normalmente, para controlar los bucles necesitamos una variable auxiliar (variable local) para controlar las iteraciones.

En los bucles REPEAT el bloque de sentencias se ejecuta por lo menos una vez. Después, se evalúa la condición.

Si la evaluación de la condición es FALSE, se vuelve a repetir la ejecución del bloque de sentencias.

```
REPETIR

        lista de declaraciones
HASTA condición de búsqueda
FIN DE LA REPETICIÓN
```

Ejemplo2: Procedimiento que dado un número entero positivo cree una cadena de caracteres con la cadena 'HOLA' repetida tantas veces como indique el número. El procedimiento también mostrará el resultado mediante sentencia SELECT.

```
delimitador //

CREAR PROCEDIMIENTO ejemplo2(num int sin signo)
comenzar
        declare Contador int default 0; -- Para controlar las iteraciones
        declare cadena varchar(250) default ''; -- Per guardar el resultat
        REPETIR

                SET cadena= CONCAT(cadena,'HOLA ');

                SET Contador=Contador+1; --Cambia la variable del bucle

        UNTIL Contador>=Número END REPEAT;
        SELECCIONAR             cadena;
fin//


delimitador ;
llamar a ejemplo2(4);
```

Este algoritmo es una buena respuesta al problema para empezar; pero fíjate: si el número pasado como parámetro fuera 0, la cadena resultante sería ' HOLA' en lugar de la cadena vacía que sería lo deseable.

```
llamar a exemple2(0);
```

Esto ocurre porque el bloque de sentencias se ejecuta al menos una vez en los bucles REPEADO. Podríamos solucionarlo con un IF de forma que sólo se accediera al bucle si el número es mayor que 0.

```
delimitador //

CREAR PROCEDIMIENTO ejemplo2bis(num int sin signo)

comenzar

        declare Contador int default 0; -- Para controlar las iteraciones

        declare cadena varchar(250) default ''; -- Per guardar el resultat

        SI num>0 ENTONCES

                REPETIR

                        SET cadena= CONCAT(cadena,'HOLA ');

                        SET Contador=Contador+1;

                UNTIL Contador>=Número END REPEAT;

        FIN SI;

        SELECCIONAR             cadena;

fin//


delimitador ;

llamar a exemple2bis(0);
```

Sentencia WHILE

En los bucles WHILE el bloque de sentencias se ejecuta sólo si la condición se evalúa a

TRUE. Puede darse el caso de que el blog no se ejecute ninguna vez.

La ejecución del bloque de sentencias (iteración) se repite MIENTRAS la evaluación de la condición

Sigue VERDADERO.

Cuando en una iteración la condición se evalúa en FALSE, el flujo de la rutina continúa por la siguiente sentencia después del END WHILE.

```
MIENTRAS condición_de_búsqueda HACER

    lista de declaraciones

TERMINAR MIENTRAS
```

Ejemplo3: Implementaremos el mismo supuesto que en el Ejemplo2 con un bucle WHILE. En este caso, no será necesario utilizar un IF para eliminar la anomalía cuando el valor del parámetro es 0. Si el parámetro tuviera valor 0, no se cumpliría la condición del WHILE y no se llegaría a ejecutar su código; con lo que la variable cadena tendría el valor con la que l'hem inicialitzada: declare cadena varchar(250) default ''

```
delimitador //

CREAR PROCEDIMIENTO ejemplo3(num int sin signo)
comenzar
        declare Contador int default 0; -- Para controlar las iteraciones
        declare cadena varchar(250) default ''; -- Per guardar el resultat
        WHILE Contador< num DO
                SET cadena= CONCAT(cadena,'HOLA ');
                SET Contador=Contador+1; --Cambia la variable del bucle
        TERMINAR MIENTRAS;
        SELECCIONAR cadena;
fin//


delimitador ;
llamar a ejemplo3(4);
```

Funciones

Las funciones son rutinas o subprogramas compuestos por un conjunto de sentencias, agrupadas lógicamente que se guardan en la base de datos y que se ejecutan como una unidad. El objetivo principal de la función es devolver un resultado.

Una función puede tener cero o muchos parámetros de entrada, sólo de entrada (IN), y siempre retorna un valor.

Una función se puede utilizar como el resultado que devuelve. Es decir, podemos invocarla para:

- Asignar un valor a una variable.
- Dentro de una sentencia SELECT
- Dentro de una expresión.

Sintaxis

```
CREAR FUNCIÓN sp_name ([func_parameter[,...]]) DEVUELVE tipo
  [característica ...] rutina_cuerpo

parámetro_función:
  tipo param_name

tipo:
  Cualquier tipo de datos MySQL válido

característica:
  COMENTARIO 'string'
  | LENGUAJE SQL
  | [NO] DETERMINISTA
  | { CONTIENE SQL | NO SQL | LEE DATOS SQL | MODIFICA DATOS SQL }
  | SEGURIDAD SQL { DEFINIDOR | INVOCADOR }

cuerpo_de_rutina:
  Valid SQL routine statement (como mínimo una sentencia RETURN)
```

Puedes encontrar más información en la documentación oficial de MySQL.

Parámetros de entrada.

En una función todos los parámetros son de entrada, por tanto, no será necesario utilizar la palabra reservada IN frente al nombre de los parámetros.

Ejemplo cabecera de una función: cabecera de la función contar_productos que tiene un parámetro de entrada llamado gamma y devuelve un entero.

```
CREATE FUNCTION contar_productos(gama VARCHAR(50)) RETURNS INT
```

Resultado de salida.

Una función siempre devolverá un valor de salida asociado al nombre de la función. En la definición de la cabecera de la función es necesario definir el tipo de dato que devuelve con la palabra reservada RETURNOS y en el cuerpo de la función debemos incluir la palabra reservada RETURN para devolver el valor de la función.

Una función puede tener varias sentencias RETURN. Hay que tener en cuenta que la ejecución de RETURNO implica la salida de la función.

Ejemplo 4: Función que dado como parámetro el nombre de una gama devuelva el número de productos de esa gama que hay en la BD (tabla Productos).

En este ejemplo se muestra un ejemplo de la estructura de una función en la que se puede ver el uso de las palabras reservadas RETURNS y RETURN.

```
USE Jardineria;
DELIMITADOR $$
CREATE FUNCTION comptar_productes(Pgamma VARCHAR(50))RETURNS INT UNSIGNED
COMENZAR
        DECLARAR num int;

        SELECCIONAR count(*) EN num DE Productos
        DONDE Gama=Pgamma;

        DEVOLVER num;
FIN
$$
```

Ya hemos creado la función. Ahora para 'usarla', podemos hacerlo como haríamos con un valor como el que devuelve (en este caso un entero). Por ejemplo: podríamos hacer cualquiera de estas invocaciones:

```
delimitador;

siete @a=contar_productos('Aromáticas');
siete @b=100*contar_productos('Frutales');

select 2+contar_productos('Ornamentales'), @a, @b;
```

Características de la función

Después de la definición del tipo de dato que devolverá la función con la palabra reservada RETURNOS, debemos indicar las características de la función. Las opciones disponibles son las siguientes:

- DETERMINISTICO: Indica que la función siempre devuelve el mismo resultado cuando se utilizan los mismos parámetros de entrada.
- NOTE DETERMINISTICO: Indica que la función no siempre devuelve el mismo resultado, aunque se utilizan los mismos parámetros de entrada. Ésta es la opción que se selecciona per defecte quan no s'indica una característicade manera explícita.
- CONTAINS SQL: Indica que la función contiene sentencias SQL, pero no contiene sentencias de manipulación de datos. Algunos ejemplos de sentencias SQL que pueden aparecer en este caso son operaciones con variables (Ej: SET @x = 1) o uso de funciones de MySQL

(Ej: SELECT NOW();) entre otros. Pero en ningún caso aparecerán sentencias de escritura o lectura de datos.

- NO SQL: Indica que la función no contiene sentencias SQL.
- READS SQL FECHA: Indica que la función no modifica los datos de la base de datos y que contiene sentencias de lectura de datos, como la sentencia SELECT.
- MODIFÍAS SQL FECHA: Indica que la función sí modifica los datos de la base de datos y que contiene sentencias como INSERT, UPDATE o DELETE.

Para poder crear una función en MySQL es necesario indicar al menos una de estas tres características:

- DETERMINISTA
- SIN SQL
- LEE DATOS SQL

Si no se indica al menos una de estas características obtendremos el siguiente mensaje de error..

```
ERROR 1418 (HY000): Esta función no tiene ningún carácter DETERMINISTA, NO
SQL, o LEE DATOS SQL en su declaración y el registro binario está habilitado (es posible
que desee utilizar la opción menos segura).
variable_creadores_de_funciones_de_confianza_log_bin)
```

Es posible establecer el valor de la variable global log_bin_trust_function_creators en 1, para indicar a MySQL que queremos eliminar la restricción de indicar alguna de las características anteriores al definir una función almacenada. Esta variable está configurada con el valor 0 por defecto y para poder modificarla es necesario contar con el privilegio SUPER.

```
ESTABLECER GLOBAL log_bin_trust_function_creators = 1;
```

MySQL :: Manual de referencia de MySQL 5.7 :: 23.7 Registro binario de programas almacenados

EJERCICIOS

1. Escribe una función que reciba un número entero de entrada y devuelva PAR si el número es par o IMPAR de lo contrario. (El operador MOD devuelve el resto de la división entera entre dos números. Así, por ejemplo, el resultado de la operación 26 MOD 3 es 2 (queda dividir 26 entre 3)

2. Escribe una función que devuelva el valor de la hipotenusa de un triángulo a partir de los valores de sus catetos.

3. Escribe una función que reciba como parámetro de entrada un valor numérico que represente un día de la semana y que devuelva una cadena de caracteres con el nombre del día de la semana correspondiente. Por ejemplo, para el valor de entrada 1 debería devolver la cadena lunes.

4. Escribe una función que reciba tres números reales como parámetros de entrada y devuelva el mayor de los tres.

5. Escribe una función que devuelva como salida el número de años que han transcurrido entre dos fechas que se reciben como parámetros de entrada. Por ejemplo, si pasamos como parámetros de entrada las fechas '2018-01-01' y '2008-01-01' la función debe devolver que han pasado 10 años.
   Para realizar esta función puede hacer uso de las siguientes funciones que nos proporciona MySQL:
   - DATEDIFF
   - ABS
   - TRUNCAR

6. Escribe una función que devuelva el número de Pedidos que se ha hecho en un año pasado como a parámetro.

7. Usando solo el operador suma (+). Función que dados dos números enteros sin signo (enteros positivos) pasados como parámetros, devuelva el resultado de multiplicarlos.

8. Función que dado un entero sin signo calcule (retorne) su factorial.